10.84
l6r
o.884

Math

8

Report No. UIUCDCS-R-77-884

UILU-ENG 77 1740

AN IMPLICIT ENUMERATION PROGRAM FOR ZERO-ONE INTEGER PROGRAMMING - ILLIP-2

by

Ming Huei Young

June 1977

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

Report No. UIUCDCS-R-77-884

AN IMPLICIT ENUMERATION PROGRAM FOR ZERO-ONE INTEGER PROGRAMMING - ILLIP-2

by

Ming Huei Young

June 1977

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

is work was supported in part by the National Science Foundation under Grant . DCR73-03421 and was submitted in partial fulfillment of the requirements r the degree of Master of Science in Computer Science in the Graduage College the University of Illinois at Urbana-Champaign, 1976.

ACKNOWLEDGEMENT

The author wishes to express his gratitude to his advisor, Professor S. Muroga. His helpful advising in preparing of this thesis, careful reading and correcting of the original manuscript are most appreciated.

Discussion with Dr. T.K. Liu of his program of ILLIP is acknowledged.

The author would also thank his family's support during his study in University of Illinois.

This work was supported in part by the Department of Computer Science, University of Illinois and also the National Science Foundation under Grant No. DCR 73-03421.

Mrs. Arbatsky's nice typing is also appreciated.

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION

The integer linear programming problem can be stated as:

$$\begin{aligned} &\text{minimize} && \vec{c} \cdot \vec{x} \\ &\text{subject to} && \vec{b} + \vec{A} \cdot \vec{x} \geq \vec{0} \end{aligned}$$

where $\vec{c}$ is a vector of n non-negative coefficients, $\vec{b}$ is a column vector of m constraints, A is an m $\times$ n matrix and $\vec{x}$ is a column vector of n integer variables. When variables are restricted to zero or one, the problem is called the zero-one linear programming problem.

Many problems such as logical network design problems, traveling-salesman problems, flight-scheduling problems and set covering problems can all be formulated into zero-one linear programming problems [1].

The implicit enumeration algorithm, developed and improved by Balas [2], Glover [3], Geoffrion [4], Ibaraki, Liu, Baugh and Muroga [5], is a very powerful algorithm for solving the zero-one linear programming problem.

A computer program package named ILLIP was developed based on the algorithm, incorporating improved procedures discussed in [5] such as "pseudo-underlining". Since its completion in 1968, ILLIP has been used by many people inside and outside the University of Illinois. In this paper, ILLIP-2, a revision of ILLIP is presented. ILLIP-2 adds new options that

(1) find more than one optimal solution,

(2) print feasible solutions found during the executioin,

(3) at a user's specification, it can skip checking underlining and pseudo-underlining conditions,

(4) find a suboptimal solution which has a specified deviation from an optimal value (only one suboptimal solution).

Some minor mistakes in ILLIP are also corrected. Although ILLIP is available in FORTRAN, the ILLIP-2 is available in both FORTRAN and PL/I.

This report describes new features added in the ILLIP-2 and how the ILLIP is modified. The features of the FORTRAN and the PL/I versions are compared. Some computational examples by ILLIP and ILLIP-2 are presented. Computational result shows that ILLIP-2 is more efficient than ILLIP. Also more error messages will be printed when the input data prepared by a user are not in correct form.

Integer programming with non-linear objective function or non-linear constraints can be converted into zero-one linear problems. Conversion methods are discussed in [12].

Reader who wants to modify the program in order to more efficiently solve their problems can get sufficient information from [12].

CHAPTER 2 NEW FEATURES IN ILLIP-2

Four new features have been added in ILLIP-2.

2.1 Alternative Optimal Solutions.

The first feature added is that a user can specify a maximum number of alternative optimal solutions he wants. The user provides two parameters to the program; one indicates that more than one optimal solution is desired and the other parameter indicates the number of alternative optimal solutions that he needs. When the number of alternative optimal solutions is greater than the number specified, only the specified number of alternative optimal solutions is printed. When the number of alternative optimal solutions found is less than or equal to the number specified, only those that are found are printed.

In order that this feature be implemented, some subroutines in ILLIP are modified and two subroutines, NEWSOL and STRSOL, are added to the main subroutine CHK-IEQ (Check-Inequality).

(1) NEWSOL

When the CHK-IEQ routine finds the first feasible solution, the CHK-IEQ routine will call subroutine NEWSOL to clear all the previous best feasible solutions that have been found and to store this new feasible solution as the best feasible solution.

(2) STRSOL

When the CHK-IEQ routine finds a feasible solution which has the same objective value as the best feasible solution that has been found, the CHK-IEQ routine will call subroutine STRSOL to store this new feasible solution along with the best feasible solutions found so far.

Incorporating these two subroutines, the CHK-IEQ of ILLIP procedure is modified as shown inside the dotted line square in Figure 2.1.

In order to facilitate finding more than one optimal solution, some modifications were made in the algorithm stated in [6], as follows.

1. The inequality ZMAX-1 - $c \cdot x \geq 0$ that is augmented to the constraints is changed to ZMAX-$\vec{c} \cdot \vec{x} \geq 0$.
2. When a new feasible solution is found in CHK-IEQ, this solution has to be tested whether it is better than the solution in the incumbent.
3. The underlining condition [5]:

   "When $a_{ij} \leq 0$ for all i such that $\ell_i(s) < 0$, $x_j = 0$ can be immediately added with an underline to the current partial solution."

   is changed to

   "When $a_{ij} \leq 0$ for all i such that $\ell_i(s) < 0$ and $c_j > 0$, $x_j = 0$ can be immediately added with an underline to the current partial solution".

   This is because if $c_j = 0$, then there may exist some feasible completion of {s,j} which has the same objective value as the best feasible completion of {s,-j} has. Thus this change is necessary when we want more than one optimal solution.
4. Let Q be the set of the indices of the fixed variables with value 1. Let NCX = $\sum_{i \in Q} c_i$. When some free variable x_j not in Q is forced to 1 and NCX + c_j equals the best value $\bar{z}$ found so far, there may exist some completion of {s,j} which has the objective function value $\bar{z}$. So we do not backtrack when more than one alternative optimal solution is desired.

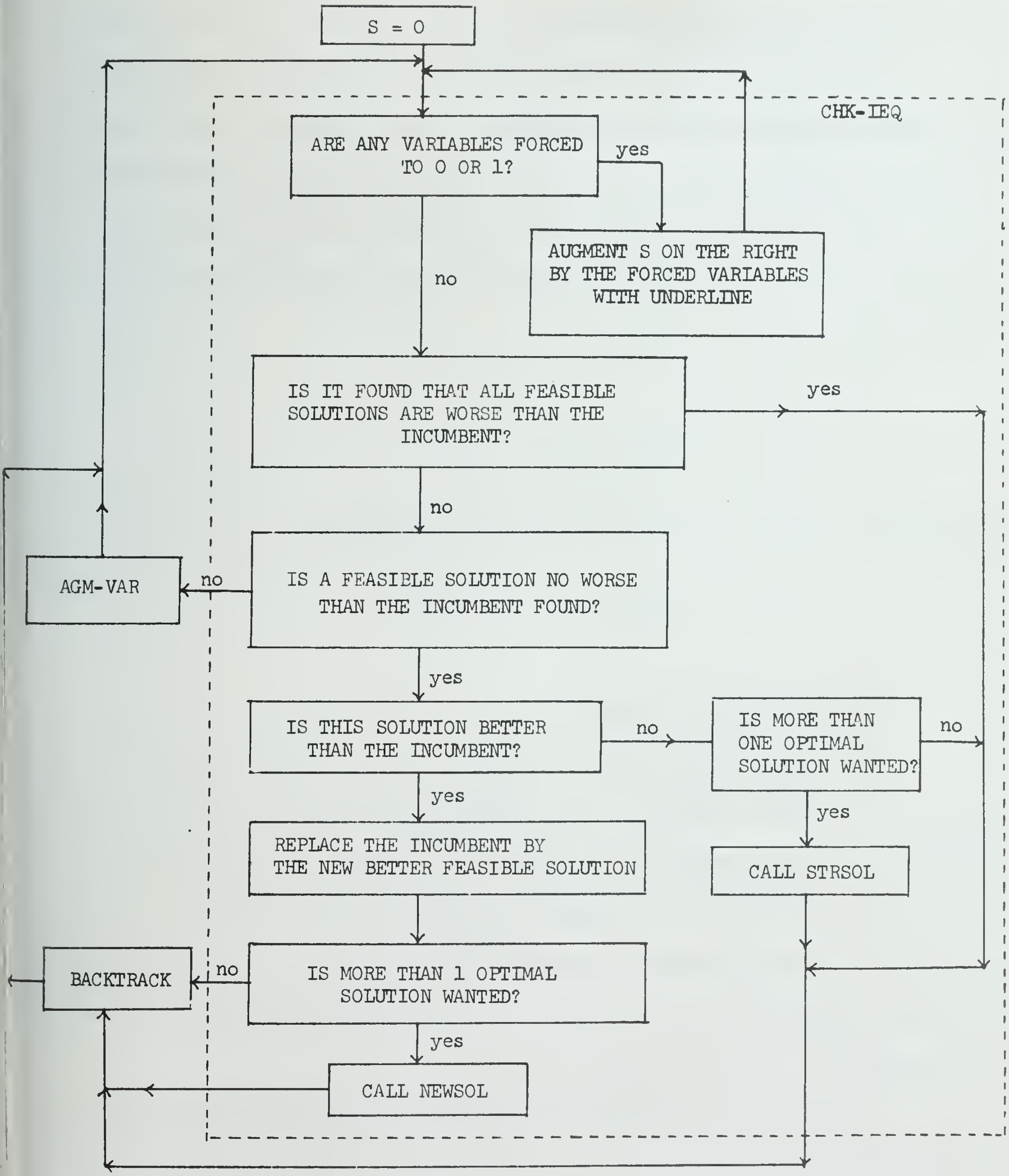
S = 0
CHK-IEQ
ARE ANY VARIABLES FORCED TO 0 OR 1?
yes
AUGMENT S ON THE RIGHT BY THE FORCED VARIABLES WITH UNDERLINE
no
IS IT FOUND THAT ALL FEASIBLE SOLUTIONS ARE WORSE THAN THE INCUMBENT?
yes
no
AGM-VAR
no
IS A FEASIBLE SOLUTION NO WORSE THAN THE INCUMBENT FOUND?
yes
IS THIS SOLUTION BETTER THAN THE INCUMBENT?
no
IS MORE THAN ONE OPTIMAL SOLUTION WANTED?
no
yes
yes
REPLACE THE INCUMBENT BY THE NEW BETTER FEASIBLE SOLUTION
CALL STRSOL
BACKTRACK
no
IS MORE THAN 1 OPTIMAL SOLUTION WANTED?
yes
CALL NEWSOL

Fig. 2.1 Flow charts of the modified CHK-IEQ.

2.2 Solutions with Deviated Value.

The second feature added in ILLIP-2 is that a user can specify an acceptable deviation DVSN from an optimal value. When the deviation specified is 0, the solution obtained is an optimal solution. When the deviation specified is equal to or greater than 1, the solution obtained is only a suboptimal solution with value $\bar{z}$ satisfying $\bar{z} - z^* \leq \text{DVSN}$, where z^* is the optimal value of this problem.

In order to do this, the inequality

$$\text{ZMAX-DVSN} - \vec{c} \cdot \vec{x} \geq 0$$

is augmented to the constraints $\vec{b} + A\vec{x} \geq 0$ instead of

$$\text{ZMAX-1} - \vec{c} \cdot \vec{x} \geq 0$$

2.3 Printing of feasible solutions.

The third feature added in ILLIP-2 is that a user can specify that some of the feasible solutions found during the execution of the program be printed. The user provides the program three parameters. One specifies this option and the other two parameters specify which feasible solutions are wanted. When the feasible solutions specified are found, they are printed with their objective function values. How to specify this option is described in the manual of ILLIP-2 in [12].

To implement this feature we only need one subroutine to print feasible solutions found.

2.4 Skip Of Checking Underlining And Pseudo-Underlining Conditions.

For some problems (e.g., logical network design problems and traveling salesman problems) pseudo-underlining and underlining conditions are seldom satisfied; then the time spent for checking these conditions is wasted. The fourth feature added in ILLIP-2 is that a user can specify whether the underlining and pseudo-underlining conditions are to be checked or not. The input specifications of this option are described in the manual of ILLIP-2 [12].

In order to do this a special checking routine is added in ILLIP-2. (See the flow charts in [12].) Execution results show that, for logical network design problems, the use of this special routine has reduced computation time by roughly 10% on the average.

CHAPTER 3 MODIFICATIONS OF ILLIP

Some other modifications have also been made to ILLIP.

3.1 Input Of Inequalities.

The input of inequalities has been simplified even though the input format of ILLIP is not changed. By this change the core storage requirement has been reduced by at least 4 K bytes and the execution time is reduced.

ILLIP first reads inequalities into an array variable INPT of length 2000 and then transfers these inequalities to array variables RWCF, COL, RWPT, RWLT (or CLCF, ROW, CLPT, CLLT) which store the inequalities in the program (the program is shown in [12]). In ILLIP-2, the inequalities are read directly into variables RWCF, COL, RWPT, RWLT (or CLCF, ROW, CLPT, CLLT). 4 K bytes storage for variable INPT is saved.

3.2 Set-Up Of Initial Partial Solution.

Subroutine SINTL, which reads in a partial solution is slightly modified so that it can accept a partial solution with some variables pseudo-underlined. The original ILLIP can not accept a partial solution with variable pseudo-underlined. This is inconvenient when we need to provide some partial solution with variables pseudo-underlined as an initial partial solution.

3.3 Initial Upper Bound.

If initial upper bound ZMAX supplied by a user is exactly equal to the optimal value of the problem, ILLIP can not detect this optimal solutio and will find the problem to be infeasible. The program is modified so that

ILLIP-2 can detect an optimal solution when the initial upper bound supplied is equal to the optimal value of the problem.

3.4 Check Of Input Sequence.

ILLIP-2 will check the increasing column index sequence of each row (or the increasing row index sequence of each column) in the input of inequalities. If the sequence is not in increasing order for each row (or column), the program will give a message to a user and stop the execution, since an incorrect sequence of the input of the inequalities will supply the program inequalities different from those of the problem. ILLIP does not check the increasing row sequence (or column sequence). If the inequalities are not prepared as stated in [7] (i.e., column sequences or row sequences are not in increasing order), the result obtained from ILLIP will not be a correct solution.

3.5 Check Of Augmenting Scheme Number.

When a scheme number supplied by a user is different from 1, 2, or 3, ILLIP-2 will assume the scheme number to be 1, give an error message and continue the execution instead of halting execution in ILLIP.

3.6 Inequality-Checking Procedure.

The checking procedure used in ILLIP is line checking, i.e., all the inequalities are checked one by one from the first inequality to the last inequality. This is repeated until no more new free variables are forced to 0 or 1. In ILLIP-2 (FORTRAN version) the chain-checking procedure discussed in [5] is used, i.e., all the inequalities that must be checked are chained together and only these inequalities are checked. When a new condition occurs, some inequalities not in the chain may need be checked again and are added to the chain. When an inequality is checked, it is deleted from the chain. By this way of checking, the time used in exammining which inequality needs to be checked is saved. For logical network design problems, the use of chain-checking procedure reduces the computation time by 30%.

3.7 Output Printing.

The print-out routine in ILLIP is modified such that it can print additional alternative optimal solutions when more than one optimal solution is desired.

3.8 Card Output.

The routine SOLPNH in ILLIP that punches out a partial solution is modified such that it also punches out the best feasible solution found so far, if any, and also alternative best feasible solutions if more than one optimal solution is desired. All these solutions and related information which are punched will be used for resuming the next run, when the computation is interrupted.

3.9 Reading-in Of Solutions.

The routine SINTL in ILLIP that reads in a partial solution is modified such that it not only reads in a partial solution but also the best feasible solution found so far, if any, and also alternative best feasible solutions if more than one optimal solution is desired.

3.10 Printing The Number Of Feasible Solutions.

The total number of feasible solutions found during the execution will be counted and will be printed at the end of the execution or at every finite number of iterations specified by a user.

3.11 Optimization Of Source Program.

The source program of ILLIP is optimized in ILLIP-2 so that the execution efficiency is improved by 5%.

3.12 Core Storage Occupied By Variables.

Since each PS(I) (NS(I), and YS(I)) occupies only 2 bytes in ILLIP, it will be overflow when coefficients of inequalities are too large. It is modified in ILLIP-2 that each PS(I) (NS(I), and YS(I)) occupies 4 bytes.

3.13 Augmentation Schemes.

The following is not a modification but it is pointed out that Scheme 3 of AGMT-VAR used in ILLIP is not the same as the one stated in [5], [6], [7]. Scheme 3 used in ILLIP is

selection of a free variable x_{j_0} which maximizes

$$\sum_{i=0}^{m} \min \{y_i(s) + a_{ij}, 0\} + \sum_{i \in u} a_{ij}$$

among all free variables x_j, where

$$u = \{i | y_i(s) \geq 0, y_i(s) + a_{ij} < 0\} .$$

It is different from what is stated in [5], [6], [7]:

selection of a free variable x_{j_0} which maximizes

$$\sum_{j=0}^{m} \min \{y_i(s) + a_{ij}, 0\}$$

among all free variables x_j.

Computational experiments show that Scheme 3 used in ILLIP is more efficient than Scheme 3 stated in [5], [6], [7] (by 30% to 60% for some examples in Chapter 4).

CHAPTER 4 COMPARISON OF THE FORTRAN AND PL/I VERSIONS OF ILLIP-2.

The PL/I version of ILLIP-2 differs from the FORTRAN version in input format, storage allocation, execution efficiency and procedure of checking inequalities.

(1) Input format.

In the FORTRAN version the input format is fixed, i.e., each input data has to be punched in some fixed input fields specified by the program. This will take a user more time to prepare this input data.

In the PL/I version the input format is not as fixed as that of the FORTRAN version and can be more easily prepared.*

(2) Storage allocation.

In the FORTRAN version, the length of each array variable is fixed. In order that this program can be used to solve big problems (i.e., a large number of variables and a large number of inequalities), each array variable is assigned a sufficiently large length. The user of this version can not change it unless the program itself is modified. When a small problem is to be solved by this version, a lot of core storage for the big array is wasted.

The lengths of array variables in the PL/I version are to be specified by a user at the beginning of the execution of the program. When a small problem is to be solved by the PL/I version, the user can claim a small array length for each variable and only a small amount of storage is then used.

---

* See Chapter 5 of the Manual of ILLIP-2 [12].

(3) Execution efficiency.

Computational results show that the FORTRAN version is more efficient than the PL/I version. Some problems solved by the FORTRAN version take one third to one fifth of the time it will take when solved by the PL/I version. The difference is caused by the difference in the compiler efficiencies of the different languages; i.e., the code generated by the FORTRAN compiler is more efficient than the codes generated by the PL/I compiler.

(4) Checking procedure of inequalities.

The inequality checking procedure used in the PL/I version is a sequential checking and the checking procedure used in the FORTRAN version is a chain checking. This is because large scale problems would be solved by the FORTRAN version only and there is no point to speed up the execution of the PL/I version.

CHAPTER 5 SOME COMPUTATIONAL RESULTS

In the following tables, Tables 5.1 through 5.5, each number in each column under 'sec.' shows the time (in seconds) used, and each number in each column under 'ITER.' shows the number of iterations for solving each problem. The computational results are obtained by running the program on an IBM 360/75I. The compilers used are FORTRAN H with option 2 and PLX. The time indicated in each table includes the initialization time, I/O time, and computation time. Entry ' - ' in each table shows that the test was not made.

(1) General zero-one linear programming problem.

Several general zero-one linear programming problems with different sizes were tested. The coefficients of each problem are randomly generated. The computational results are shown in Table 5.1.

(2) Generalized set covering problems.

A generalized set covering problem is:

$$\begin{aligned} &\text{minimize} && \sum_{j=1}^{n} c_j x_j && \\ &\text{subject to} && \sum_{j=1}^{n} a_{ij} x_j \geq 1, && \text{for } i = 1,2,\dots,m, \\ & && x_j = 0 \text{ or } 1 && \text{for } j = 1,2,\dots,n, \\ &\text{where} && a_{ij} = 0 \text{ or } 1 && \text{for } i = 1,2,\dots,m, \\ & && && \phantom{\text{for }} j = 1,2,\dots,n, \end{aligned}$$

and c_j's are non-negative integers.

Two such problems with different sizes are tested. The coefficients a_{ij} are randomly generated with non-zero density of 0.10. The computational results are shown in Table 5.2.

Table 5.1 Computational results of general zero-one linear programming problem.

			FORTRAN VERSION						PL/I VER.	
NO. OF VAR.	NO. OF INEQ.	SCHEME	ILLIP		ILLIP-2 SINGLE SOL.		ILLIP-2 MULTIPLE SOL.		ILLIP-2 SINGLE SOL	
			SEC.	ITER.	SEC.	ITER.	SEC.	ITER.	SEC.	ITER
15	35	1	2.17	488	1.86	488	1.80	495	-	-
		2	2.70	488	1.78	488	1.77	495	-	-
		3	2.25	488	1.73	488	1.82	495	8.37	488
		4*	-	-	3.42	718	5.20	1193	-	-
12	4	1	0.47	25	0.35	25	0.35	25	-	-
		2	0.40	21	0.32	21	0.38	22	-	-
		3	0.46	16	0.30	16	0.36	18	0.92**	18*
		4*	-	-	0.49	23	0.49	23	-	-
15	15	1	1.25	227	1.04	227	1.31	287	-	-
		2	1.32	227	1.18	227	1.25	287	-	-
		3	1.17	148	0.8	148	0.98	194	-	-
		4*	-	-	2.39	524	-	-	-	-
23	4	1	1.41	381	1.47	381	1.77	461	-	-
		2	1.68	384	1.52	384	1.95	465	5.75	384
		3	1.97	528	1.96	528	2.34	634	-	-
80***	12	1	-	-	(49.92)	5773	-	-	-	-
		2	-	-	(50.58)	5773	-	-	-	-
		3	-	-	(54.34)	5026	-	-	-	-

* Scheme 4 is the augment Scheme 3 as stated in [5], [6], [7].

** Multiple solutions are found.

*** This result is provided by George, Bob. The time in parentheses shows only computation time, while the time without parentheses includes the initialization, I/O time, and computation time.

Table 5.2 Computational results of generalized set-covering problems.

			FORTRAN VERSION						PL/I VER.	
O. OF VAR.	NO. OF INEQ.	SCHEME	ILLIP		ILLIP-2 SINGLE SOL.		ILLIP-2 MULTIPLE SOL.		ILLIP-2 SINGLE SOL.	
			SEC.	ITER.	SEC.	ITER.	SEC.	ITER.	SEC.	ITER.
30	15	1	1.00	91	0.81	91	1.55	281	-	-
		2	0.95	91	0.79	91	1.62	281	-	-
		3	1.10	91	0.75	91	1.62	281	6.52**	281**
		4*	-	-	1.84	341	5.52	1283	-	-
50	25	1	7.80	1703	6.82	1703	10.68	2794	-	-
		2	8.40	1703	7.43	1703	11.46	2794	-	--
		3	7.57	1703	7.14	1703	11.08	2794	32.39	1946
		4*	-	-	>19.52	>5,000	-	-	-	-

* See the footnote of Table 5.1.

** For multiple solutions.

(3) Traveling Salesman Problem.

Traveling salesman problem is:

$$\text{minimize} \quad \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_{ij}$$

subject to

$$\sum_{\substack{i=1 \\ i \neq j}}^{n} x_{ij} = 1 \quad \text{for } j = 1,2,\ldots,n \;,$$

$$\sum_{\substack{j=1 \\ j \neq i}}^{n} x_{ij} = 1 \quad \text{for } i = 1,2,\ldots,n \;,$$

$$x_{ij} = 0 \text{ or } 1 \quad \text{for } i = 1,2,\ldots,n \;,$$
$$j = 1,2,\ldots,n \;,$$

$$\text{and} \quad \sum_{i \in s} \sum_{j \in \bar{s}} x_{ij} \geq 1 \quad \text{for every } s \subset \{1,2,\ldots,n\}$$

where d_{ij}'s are given non-negative integers.

This problem is reformulated as [8]:

$$\text{minimize} \quad \sum_{i \neq j}\sum d_{ij} x_{ij}$$

subject to

$$\sum_{\substack{i=1 \\ i \neq j}}^{n} x_{ij} = 1 \;, \quad j = 1,2,\ldots,n \;,$$

$$\sum_{\substack{j=1 \\ j \neq i}}^{n} x_{ij} = 1 \;, \quad i = 1,2,\ldots,n \;,$$

$$u_i - u_j + (n-1)\, x_{ij} \leq n - 2 \;, \quad 2 \leq i \neq j \leq n \;,$$

$$x_{ij} = 0 \text{ or } 1 \quad \text{for} \quad i = 1,2,\ldots,n \ ,$$
$$j = 1,2,\ldots,n \ ,$$
$$u_i : \text{integer smaller than } n - 2, \text{ for}$$
$$i = 2,3,\ldots,n \ .$$

Each variable u_i is converted into zero-one variable as stated in Appendix. Three problems were tested. The coefficients d_{ij} were randomly chosen. The computational results were shown in Table 5.3.

(4) Logical network design problem.

This problem is to find a logic network consisting of a fixed number of NOR gates which realizes a given switching function and minimizes the number of interconnections. The formulation of this problem into zero-one linear programming problem is discussed in [9], [10], [11]. Three different switching functions of three variables were tested. The computational results are shown in Table 5.4.

(5) Comparison of two cases with and without checking underlining and pseudo-underlining conditions.

Underlining and pseudo-underlining conditions are seldom satisfied in some types of problems (e.g., logical network design problem and traveling salesman problem). For some other types of problems (e.g., generalized minimal covering problem), checking these conditions will greatly reduce computation time. Several tests have been made. The comparison is shown in Table 5.5.

Table 5.3 Computational results of traveling salesman problem

				FORTRAN VERSION						PL/I VER.	
NO. OF CITY	NO. OF VAR.	NO. OF INEQ.	SCHEME	ILLIP		ILLIP-2 SINGLE SOL.		ILLIP-2 MULTIPLE SOL.		ILLIP-2 SINGLE SOL.	
				SEC.	ITER.	SEC.	ITER.	SEC.	ITER.	SEC.	ITER.
6	45	44	1	8.65	1122	7.65	1118	12.13**	1592	-	-
			2	8.57	1078	7.45	1078	11.51	1761	32.75	1032
			3	12.07	1642	10.87	1642	12.65	2121	–	–
			4*	-	-	>38.74	>5,000	-	-	-	-
6	45	44	1	9.80	1326	8.77	1326	18.87	2735	-	-
			2	9.25	1219	8.27	1219	24.89	3488	-	-
			3	13.15	2033	12.27	2033	20.15	3661	-	-
			4*	-	-	>30.00	>4,000	>37.64	>5,000	-	-
7	60	60	3	-	-	625.72	106,118	-	-	-	-

* See the footnote of Table 5.1.

** 40 alternative optimal solutions were found; first 10 among these 40 alternative optimal solutions took 10.55 seconds and 1492 iterations, only.

Table 5.4 Computational results of logical network design problem.

NO. OF GATES	NO. OF VAR.	NO. OF INEQ.	ILLIP	ILLIP-2 (LINE CHECKING)	ILLIP-2 (CHAIN CHECKING)	ILLIP-2 (WITHOUT CHECKING UNDERLINING AND PSEUDO-UNDERLINING CONDITIONS)		BEST RESULTS OBTAINED BEFORE
						SINGLE SOL.	MULTIPLE SOL.	
6	193	417	$39.90^{sec.}$	$39.92^{sec.}$	$27.15^{sec.}$ $(20.01)^{*}$	$23.87^{sec.}$ $(16.76)^{*}$	$23.60^{sec.}$ $(16.79)^{*}$	$(17.41^{sec.})^{*}$
6	193	417	-	-	-	$(15.85^{sec.})^{*}$	$(16.78^{sec.})^{*}$	$(18.71^{sec.})^{*}$
7	256	717	-	-	-	$484.58^{sec.}$ $(468.78)^{*}$	-	$(465.69^{sec.})^{*}$

* The time in the paranthesis shows only the computational time.

Table 5.5 Comparison of two cases with or without checking underlining and pseudo-underlining conditions.

PROBLEM SIZE		WITHOUT CHECKING UNDERLINING AND PSEUDO-UNDERLINING CONDITIONS		WITH CHECKING UNDERLINING AND PSEUDO-UNDERLINING CONDITIONS		PROBLEM TYPE
NO. OF VAR.	NO. OF INEQ.	SEC.	ITER.	SEC.	ITER.	
23	4	(1.18)*	400	(1.48)*	381	General IP Problem
45	44	(6.07)*	1121	(6.62)*	1083	Traveling Salesman Problem
60	25	(10.69)*<	2,000<	(6.82)*	1703	Generalized Minimal Covering Problem
258	715	484	17,855	739	17,855	Logic Network Design Problem

* The time in the parenthesis shows only the computational time.

REFERENCES

[1] Balinski, M.L., "Integer Programming: Method, Uses, Computation," Management Science, Vol. 12, No. 3, 1965.

[2] Balas, E., "An Additive Algorithm for Solving Linear Programming With Zero-One Variables," Operations Research, Vol. 13, No. 4, pp. 517-544, 1965.

[3] Glover, F., "A Multiple Phase-Dual Algorithm for the Zero-One Integer Programming Problem," Operations Research, Vol. 13, No. 6, pp. 879-919, 1965.

[4] Geoffrion, A.M., "Integer Programming by Implicit Enumeration and Balas' Method," SIAM Review, Vol. 9, No. 2, pp. 178-190, 1967.

[5] Ibaraki, T., T.K. Liu, C.R. Baugh and S. Muroga, "An Implicit Enumeration Program for Zero-One Integer Programming," International Journal of Computer and Information Science, Vol. 1, pp. 1090-1096, 1972.

[6] Ibaraki, T., T.K. Liu, C.R. Baugh and S. Muroga, "An Implicit Enumeration Program for Zero-One Integer Programming," Report No. 305, Dept. of Computer Science, Univ. of Ill.

[7] Liu, T.K., "A Code for Zero-One Integer Linear Programming by Implicit Enumeration," Report No. 302, Dept. of Computer Science, Univ. of Ill., 1968.

[8] Miller, C.E., A.W. Tucker and R.A. Zemlin, "Integer Programming Formulation of Traveling Salesman Problems," Journal of ACM, Vol. 7, pp. 326-329, 1960.

9] Muroga, S. and T. Ibaraki, "Logical Design of an Optimum Network by Integer Linear Programming - Part I," Report No. 264, Dept. of Comp. Sci., Univ. of Ill., July, 1968.

.0] Muroga, S. and T. Ibaraki, "Logical Design of an Optimum Network by Integer Linear Programming - Part II," Report No. 289, Dept. of Comp. Sci., Univ. of Ill., Dec. 1968.

.1] Muroga, S. and T. Ibaraki, "Design of Optimal Switching Networks by Integer Programming," IEEE TC, pp. 573-582, June, 1972.

2] Young, M.H., T.K. Liu, C.R. Baugh and S. Muroga, "A Code for Zero-One Integer Programming ILLIP-2," UIUCDCS-R-77-858, Dept. of Comp. Sci., Univ. of Ill., April, 1977.

LIOGRAPHIC DATA ET	1. Report No. UIUCDCS-R-77-884	2.	3. Recipient's Accession No.
itle and Subtitle AN IMPLICIT ENUMERATION PROGRAM FOR ZERO-ONE INTEGER PROGRAMMING - ILLIP-2			5. Report Date July 1, 1977
			6.
uthor(s) Ming Huei Young			8. Performing Organization Rept. No. UIUCDCS-R-77-884
erforming Organization Name and Address Department of Computer Science University of Illinois at Urbana-Champaign Urbana, Illinois 61801			10. Project/Task/Work Unit No.
			11. Contract/Grant No. NSF DCR73-03421
ponsoring Organization Name and Address National Science Foundation 1800 G Street, N.W. Washington, D.C. 20550			13. Type of Report & Period Covered Technical
			14.

upplementary Notes

bstracts

ILLIP-2 is an improved version of the program ILLIP. This report describes new features added in the ILLIP-2 and how the ILLIP is modified. Comparison of some computational results between the ILLIP-2 and the ILLIP are shown.

y Words and Document Analysis. 17a. Descriptors

ero-one linear programming, implicit enumeration method, partial solution, nderlining, pseudo-underlining, incumbent, backtracking, optimization.

entifiers/Open-Ended Terms

)SATI Field/Group

ilability Statement elease unlimited	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages 27
	20. Security Class (This Page UNCLASSIFIED	22. Price